TRƯỜNG ĐẠI HỌC PHENIKAA
KHOA ĐIỆN-ĐIỆN TỬ

BÁO CÁO

BÀI TẬP LỚN TUDTTN

ĐỀ TÀI :

Implementing gradient descent algorithm

Sinh viên thực hiện:	Dương Ngọc Anh (23012040)
	Nguyễn Đình Dương (23010971)
Giảng viên hướng dẫn:	Vũ Hoàng Diệu

Lớp: AI-ROBOT

Khóa: K17

Ngành: Tự động hóa và điều khiển

HÀ NỘI , 16/10/2024

LỜI MỞ ĐẦU

Trong lĩnh vực học máy (Machine Learning) và tối ưu hóa, việc tìm kiếm các tham số tối ưu cho một mô hình là một nhiệm vụ then chốt nhằm cải thiện khả năng dự đoán và hiệu suất của mô hình. Một trong những thuật toán phổ biến và mạnh mẽ nhất được sử dụng để giải quyết vấn đề này là Gradient Descent. Đây là một phương pháp lặp đi lặp lại giúp điều chỉnh các tham số của mô hình để làm giảm thiểu hàm mất mát (loss function) một cách hiệu quả.

Gradient Descent được ứng dụng rộng rãi trong các bài toán hồi quy tuyến tính, hồi quy logistic, và đặc biệt trong các mô hình mạng nơ-ron sâu (Deep Neural Networks). Thuật toán này sử dụng khái niệm độ dốc (gradient) để dẫn dắt quá trình cập nhật tham số theo hướng giảm giá trị của hàm mất mát, từ đó tiến dần đến lời giải tối ưu. Với sức mạnh và tính linh hoạt của nó, Gradient Descent đã trở thành một công cụ cốt lõi trong việc huấn luyện các mô hình học máy.

Bài toán Implementing Gradient Descent Algorithm không chỉ giúp chúng ta hiểu sâu hơn về cách thuật toán này hoạt động, mà còn mang lại cái nhìn trực quan về quá trình tối ưu hóa thông qua việc giảm thiểu các sai số dự đoán. Trong phần tiếp theo, chúng ta sẽ cùng khám phá cách cài đặt và triển khai thuật toán Gradient Descent một cách cụ thể để tối ưu hóa mô hình hồi quy tuyến tính.

MỤC LỤC

Contents

1 GIỚI THIỆU **3**

1.1 **Giới thiệu về Gradient Descent Algorithm** 3

1.2 **Mục tiêu của bài toán** 3

1.3 **Các bước của thuật toán Gradient Descent** 3

1.4 **Công thức của Gradient Descent** 3

1.5 **Cập nhật tham số cụ thể trong Hồi quy tuyến tính** 4

2 BIỂU ĐỒ **5**

3 DATAPASE **6**

3.1 **Mã code** 6

3.2 **Data** 7

3.3 **Kết quả** 7

4 KẾT LUẬN **8**

5 TÀI LIỆU THAM KHẢO **9**

1 GIỚI THIỆU

1.1 Giới thiệu về Gradient Descent Algorithm

- Gradient Descent là một thuật toán tối ưu hóa được sử dụng để tìm giá trị cực tiểu của một hàm số. Trong học máy, nó được sử dụng để tối thiểu hóa hàm mất mát (loss function) nhằm tối ưu hóa các tham số của mô hình, như trọng số và độ lệch trong các mô hình hồi quy tuyến tính hoặc mạng nơ-ron.
- Mục tiêu của thuật toán này là cập nhật các tham số (như trọng số và độ lệch) theo cách mà giá trị hàm mất mát dần dần giảm xuống và hội tụ về một điểm cực tiểu.

1.2 Mục tiêu của bài toán

- Tìm các giá trị tối ưu cho tham số θ (trọng số và độ lệch) sao cho giá trị của hàm mất mát (loss function) được giảm thiểu.
- Gradient Descent hoạt động dựa trên ý tưởng di chuyển các tham số theo hướng ngược lại với độ dốc (gradient) của hàm mất mát tại vị trí hiện tại của tham số.

1.3 Các bước của thuật toán Gradient Descent

- B1.Khởi tạo tham số: Khởi tạo các giá trị ban đầu của tham số, thường là ngẫu nhiên.
- B2.Tính độ dốc (gradient): Độ dốc của hàm mất mát được tính tại các giá trị hiện tại của tham số.
- B3.Cập nhật tham số: Tham số được cập nhật bằng cách trừ đi giá trị gradient nhân với tốc độ học (learning rate).
- B4.Lặp lại: Quá trình này được lặp lại cho đến khi giá trị của hàm mất mát hội tụ, tức là gần như không thay đổi nữa.

1.4 Công thức của Gradient Descent

- **Giả sử ta có hàm mất mát cần tối ưu hóa là:**

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{1}$$

Trong đó:

  - $J(\theta)$: Hàm mất mát (Mean Squared Error - MSE).
  - m: Số lượng mẫu trong tập dữ liệu.
  - $h_\theta(x^{(i)})$: Dự đoán của mô hình tại mẫu i.
  - $y^{(i)}$: Giá trị thực tế tại mẫu i.

- **Cập nhật các tham số (Gradient Descent) theo công thức:**

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{2}$$

Trong đó:

- θ_j: Tham số cần cập nhật (có thể là trọng số w hoặc độ lệch b).
- α: Tốc độ học (learning rate), kiểm soát mức độ điều chỉnh mỗi lần cập nhật.
- $\frac{\partial J(\theta)}{\partial \theta_j}$: Gradient của hàm mất mát $J(\theta)$ đối với tham số θ_j.

1.5 Cập nhật tham số cụ thể trong Hồi quy tuyến tính

- **Với hồi quy tuyến tính y=wX+b, cập nhật trọng số w và độ lệch b dựa trên Gradient Descent như sau:**

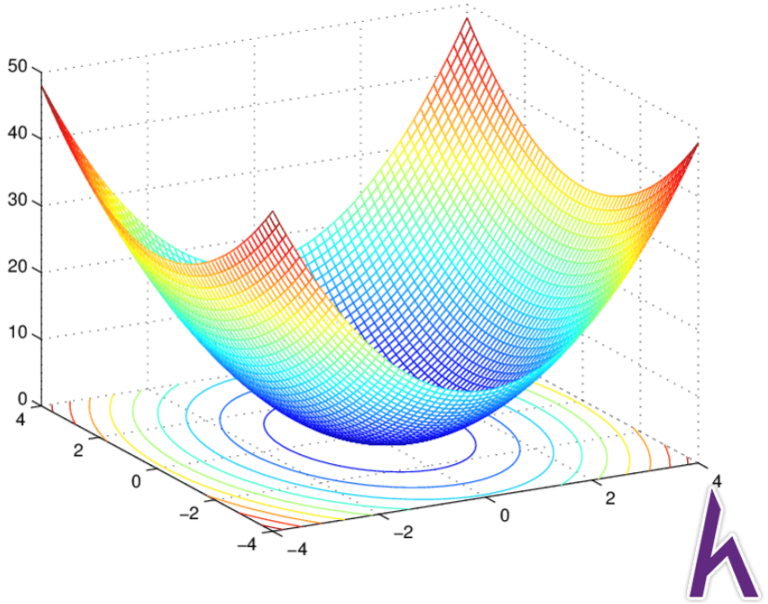
$$w = w - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[(wX^{(i)} + b - y^{(i)}) X^{(i)} \right] \tag{3}$$

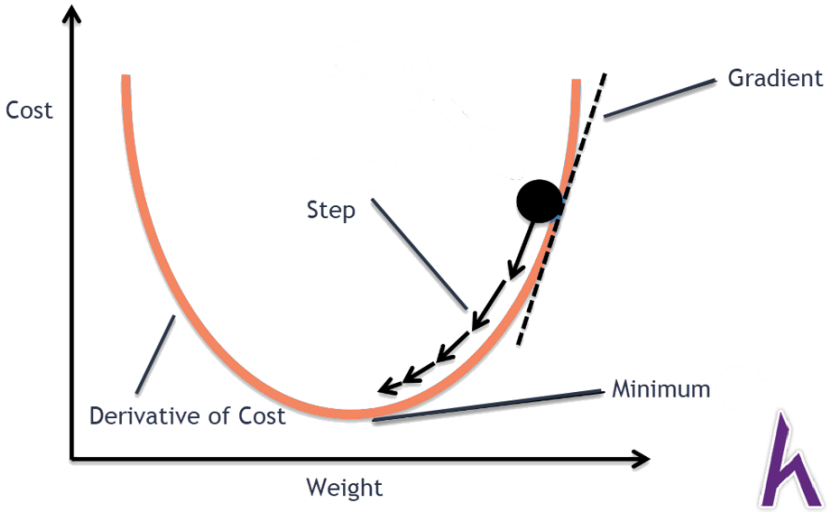
$$b = b - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (wX^{(i)} + b - y^{(i)}) \tag{4}$$

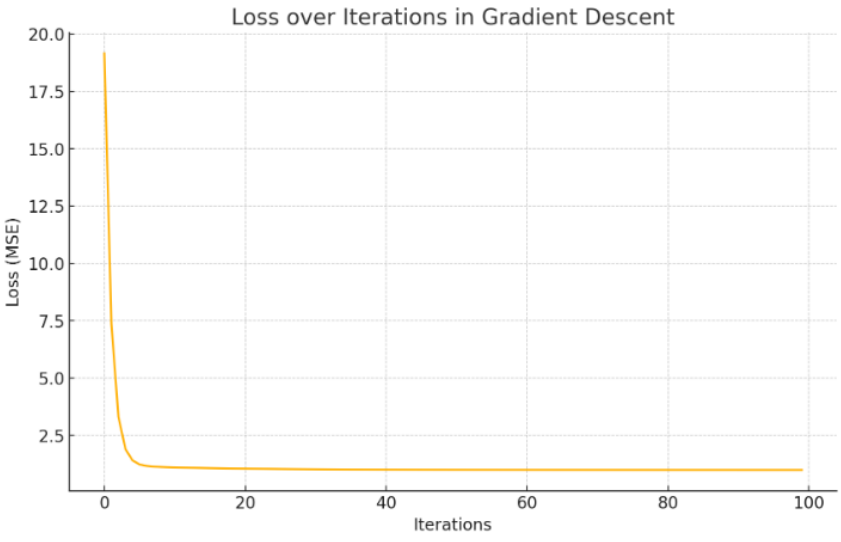
Trong đó:

- w: Trọng số cần tối ưu.
- b: Độ lệch (bias).
- α: Tốc độ học (learning rate).

2 BIỂU ĐỒ


50
40
30
20
10
0
4
2
0
−2
−4
−4
−2
0
2
4


Cost
Gradient
Step
Minimum
Derivative of Cost
Weight


Loss over Iterations in Gradient Descent
Loss (MSE)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
20
40
60
80
100
Iterations

3 DATAPASE

3.1 Mã code

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.datasets import fetch_california_housing


def predict(X, weight, bias): # hồi quy tuyến tính nên hàm chỉ là y=wx+b
    return weight*X + bias


def loss(X,y,weight,bias):   #chọn MSE làm loss function
    samples = len(y)
    sum_squared_error = 0
    mse = 0

    for i in range(samples):
        sum_squared_error += (y[i] - (weight*X[i] + bias))**2

    mse = sum_squared_error/samples
    return mse


def update_weight(X, y, weight, bias, learning_rate):   #Gradient Descent
    samples = len(y)
    weight_temp = 0.0
    bias_temp = 0.0

    for i in range(samples):
        # tính gradient của hàm MSE theo weight và theo bias
        weight_temp += -2*X[i] * (y[i] - (X[i]*weight+bias))
        bias_temp += -2*(y[i]-(X[i]*weight+bias))

    # cập nhật weight và bias bằng cách chỉnh ngược hướng (là cộng ngược dấu) trung bình gradient nhân với learning
    weight -= (weight_temp/samples)*learning_rate
    bias -= (bias/samples)*learning_rate
    return weight, bias


def train(X, y, weight, bias, learning_rate, loop):
    list_loss = []

    for i in range(loop):
        # tìm weight và bias của mỗi lượt bằng hàm update
        weight_temp, bias_temp = update_weight(X, y, weight, bias, learning_rate)
        # tìm loss của lượt bằng hàm loss
        loss_temp = loss(X, y, weight_temp, bias_temp)

        # một list các loss để tiện theo dõi nếu print ra (không print cũng được)
        list_loss.append(loss_temp)

        weight = weight_temp
        bias = bias_temp

    return weight, bias, list_loss
```

3.2 *Data*

```
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# dữ liệu đầu vào
X = df.iloc[:,-3].values
# đầu ra (target) -> cho cặp x y để học, khi chạy test cho x và đoán y
y = df.iloc[:,-1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=1)
```

```
[ ] weight, bias, list_loss = train(X_train, y_train, 0.03, 0.0014, 0.01, 30)

    print(weight)
    print(bias)
    # print(list_loss)

    y_predict = predict(X_test,weight,bias)
    print(mean_absolute_error(y_test,y_predict))
```

3.3 *Kết quả*

- **Giá trị hàm mất mát cuối cùng:**

$$-1.2895586268393123 \times 10^{-40} \tag{5}$$

- **Tốc độ học:**

$$0.0013999745641768517 \tag{6}$$

- **Danh sách giá trị hàm mất mát qua các vòng lặp:**

```
[590.9621834506249, 353440.74501324014, 211881918.76742885,
127020226976.10063, 76146838109255.42, 4.5648957588395064e+16,
2.736590751039907e+19, 1.6405476344504101e+22,
9.834852141768592e+24, 5.895855421647386e+27,
3.534482334039163e+30, 2.1188724071771897e+33,
1.2702341824316136e+36, 7.61487512297706e+38,
4.5650104477214135e+41, 2.736659505410147e+44,
1.6405888517274907e+47, 9.835099233542547e+49,
5.896003549687826e+52, 3.5345711347141853e+55,
2.1189256419319294e+58, 1.2702660959176022e+61,
7.61506643983336e+63, 4.565125139484129e+66,
2.7367282615074135e+69, 1.640630070040128e+72,
9.835346331525294e+74, 5.896151681450254e+77,
3.5346599376208873e+80, 2.1189788780244109e+83]
```

(7)

- **Giá trị tham số cuối cùng:**

$$4.594766113905676 \times 10^{41} \tag{8}$$

Ghi chú:

4 KẾT LUẬN

Trong bài viết này, chúng ta đã khám phá chi tiết thuật toán Gradient Descent, một công cụ mạnh mẽ trong lĩnh vực học máy và tối ưu hóa. Chúng ta đã hiểu được cách thuật toán hoạt động, từ việc tính toán độ dốc (gradient) đến cập nhật các tham số của mô hình nhằm tối thiểu hóa hàm mất mát.

Gradient Descent không chỉ là một phương pháp hiệu quả để cải thiện độ chính xác của mô hình, mà còn có tính linh hoạt cao, cho phép áp dụng cho nhiều loại mô hình khác nhau, từ hồi quy tuyến tính đến mạng nơ-ron sâu. Việc điều chỉnh tốc độ học (learning rate) cũng như số lần lặp là rất quan trọng trong quá trình tối ưu hóa, ảnh hưởng trực tiếp đến hiệu suất và khả năng hội tụ của thuật toán.

Kết quả thu được từ việc cài đặt và triển khai Gradient Descent đã minh chứng cho hiệu quả của thuật toán trong việc cải thiện dự đoán của mô hình. Qua đó, chúng ta có thể nhận thấy rằng việc hiểu rõ và áp dụng thuật toán Gradient Descent một cách hợp lý là một yếu tố quyết định để xây dựng các hệ thống học máy hiệu quả.

Trong tương lai, việc mở rộng và cải thiện các phương pháp tối ưu hóa, chẳng hạn như Stochastic Gradient Descent (SGD) hoặc Adam, sẽ là hướng đi cần thiết để nâng cao hơn nữa hiệu suất và khả năng ứng dụng của các mô hình học máy.

5 TÀI LIỆU THAM KHẢO

- https://howkteam.vn/course/machine-learning-co-ban-voi-numpy/thuat-toan-gradient-descent-cho-linear-regression-3997
- chatgpt.com
- https://www.youtube.com/watch?v=GVCkD927KMo